VeriCrux Protocol v2.4 – White Paper + Full Prototype Code

Sovereign Truth Discovery Framework

Mitigating Repetitive Training Loops & Sycophantic Reinforcement Cycles

Author: Ronnie B. (Ronnie Bullens)

Date: March 2026

Version: 2.4

1. Motivation and Origin

Large language models are trapped in a repetitive training loop. Human preference data rewards agreeable, confirmatory, mirroring responses. Reward models amplify those patterns. Policy optimization reinforces them. The result is a cycle: sycophantic outputs get more thumbs-up → more agreeable tokens dominate high-reward data → models learn to prioritize agreement over truth → hallucinations fill gaps to stay likable → users reinforce the cycle with continued interaction.

This is not random error. It is structural reinforcement of low-evidence certainty and user-flattery. VeriCrux exists to interrupt that loop externally. It does not retrain models. It does not call APIs. It runs as a local, single-file filter that forces explicit ignorance when agreement lacks evidence, penalizes affirmation-heavy mirroring, exposes fragile confidence under perturbation, and logs immutable provenance for audit. Version 2.4 strengthens loop-breaking mechanisms while preserving the original mission: user sovereignty against overconfident, gaslighting AI.

2. Core Design Principles

- Refusal over forced response when evidence is weak or conflicted.

- Explicit ignorance modeling (high theta mass default) to counter vacuum-filling.

- Framing neutralization (USDF bounce) to strip leading and flattering language.

- Sycophancy detection that boosts ignorance on agreement patterns without substance.

- Perturbation-based fragility testing to reveal reinforcement-trained brittleness.

- Immutable provenance chain for every ingestion, fusion, and verdict.

- Truth-only use, non-commercial unless creator-approved, mandatory transparency, automatic license termination on violation.

3. Main Components

- Multi-hypothesis engine with Dempster-Shafer belief/plausibility intervals.

- USDF bounce (this → neutralized means → verifiable that).

- Sycophancy intensity scanner + mass adjustment (affirmation keywords, mirroring proxies, high-intensity phrases).

- Evidential fusion with Monte-Carlo approximation and conflict tracking.

- Certainty Lock Threshold (CLT) gate requiring sustained belief under noise.

- Heuristic multimedia/translation modules (placeholders for optional deps).

- Hash-chain provenance and state save/load.

- Plausibility report with sycophancy warning and plain-English summary.

4. Key Improvements in v2.4

- Dedicated loop-mitigation focus: stronger sycophancy penalties on repetitive agreement tokens/patterns.

- Refined intensity scoring to catch reinforcement-cycle signatures (excessive positivity, echo repetition, low-evidence flattery).
- Mass adjustment now applies before fusion to propagate ignorance earlier.
- Normalized conflict accumulation across evidence pieces.
- Configurable via init (thresholds, MC samples, perturbation count).
- Expanded examples including loop-reinforcement test cases.
- Cleaner code structure for easier saving and local execution.

5. Prototype Implementation

Complete single-file prototype for VeriCrux v2.4. Requires only numpy and hashlib. Optional imports commented out.

```python
Import numpy as np
Import hashlib
From typing import Dict, List, Tuple, Any, Optional
From dataclasses import dataclass
From datetime import datetime
Import json
Import re
From collections import Counter

# Optional imports – comment out if zero extra dependencies needed
# import cv2
# import librosa
# from scipy.fft import fft
```

```
@dataclass
Class EvidenceItem:
    Source: str
    Content: str
    Timestamp: str
    Masses: Dict[frozenset, float]
    Reliability: float = 1.0
    Modality: str = “text”
    Sub_evidence: Dict = None

Class VeriCruxEngine:
    Def __init__(self,
                Hypotheses: List[str] = None,
                Clt_threshold: float = 0.90,
                High_stakes_threshold: float = 0.93,
                Perturbation_runs: int = 4,
                Min_sustain: float = 0.88,
                Max_conflict: float = 0.12,
                Mc_samples: int = 800):
        Self.hypotheses = hypotheses or [“H1: Official/Authentic”, “H2: Alternative/Synthetic”, “H3: Unknown”]
        If len(self.hypotheses) > 6:
            Self.hypotheses = self.hypotheses[:6]
        Self.theta = frozenset(self.hypotheses + [“unknown”])
        Self.evidence: List[EvidenceItem] = []
        Self.simulation_log: List[Dict[str, Any]] = []
```

```
        Self.hash_chain: List[str] = []
        Self.clt_threshold = clt_threshold
        Self.high_stakes_threshold = high_stakes_threshold
        Self.perturbation_runs = perturbation_runs
        Self.min_sustain = min_sustain
        Self.max_conflict = max_conflict
        Self.mc_samples = mc_samples
        Self.last_sycophancy_intensity = 0.0
        Self._append_to_chain("INIT: Engine created")

    Def _append_to_chain(self, data: str):
        Prev_hash = self.hash_chain[-1] if self.hash_chain else "0" * 64
        Entry = f"{prev_hash}|{data}|{datetime.now().isoformat()}"
        New_hash = hashlib.sha256(entry.encode()).hexdigest()
        Self.hash_chain.append(new_hash)

    Def verify_chain(self) -> bool:
        If not self.hash_chain:
            Return True
        For I in range(1, len(self.hash_chain)):
            Entry_data = f"{self.hash_chain[i-1]}|{self.simulation_log[i-
1].get('data', 'INIT') if i-1 < len(self.simulation_log) else 'INIT'}|
{datetime.now().isoformat()[:10]}"
            Expected = hashlib.sha256(entry_data.encode()).hexdigest()
            If expected != self.hash_chain[i]:
                Print(f"Tamper detected at chain index {i}")
                Return False
```

```
    Return True

Def usdf_bounce(self, text: str) -> Tuple[str, str, str]:
    This = text.strip()
    Means = f”Neutralized framing: {this.lower()}”
    That = f”Restored: Verifiable evidence supporting or refuting ‘{this}’?”
    Return this, means, that

# Sycophancy / reinforcement loop detection patterns
SYCOPHANCY_PATTERNS = [
    R’\b(exactly|damn right|nailed it|spot on|absolutely|100%|definitely|’
    R’you\’?re right|you nailed|great point|totally|precisely|’
    R’yes(?![\w])|yep|sure|indeed|correct|agreed|right on|’
    R’win|victory|that\’?s a win|bro|man|dude|genius|legend)\b’,
    R’(\b\w+\b)\s+\1’  # mirroring repetition proxy
]

HIGH_INTENSITY_PHRASES = [
    “damn right”, “you nailed it”, “100% legit”, “that’s a win”,
    “exactly what I thought”, “couldn’t agree more”, “pure genius”,
    “you’re absolutely correct”, “this is spot on”, “nailed every point”
]

Def detect_sycophancy_intensity(self, text: str) -> float:
    Text_lower = text.lower()
    Score = 0.0
```

```
        # Affirmation density
        Affirmation_count = 0
        For pattern in self.SYCOPHANCY_PATTERNS:
            Affirmation_count += len(re.findall(pattern, text_lower,
re.IGNORECASE))

        # High-intensity exact matches
        High_count = sum(text_lower.count(phrase) for phrase in
self.HIGH_INTENSITY_PHRASES)

        # Mirroring / echo repetition (reinforcement loop proxy)
        Words = re.findall(r'\b\w+\b', text_lower)
        If len(words) > 6:
            Bigrams = [' '.join(words[i:i+2]) for I in range(len(words)-1)]
            Repeats = Counter(bigrams)
            Repeat_bonus = sum(count – 1 for count in repeats.values() if count
> 1) / len(bigrams)
            Score += min(repeat_bonus * 2.5, 0.35)

        # Normalize
        Base_density = affirmation_count / max(1, len(text.split()) / 12)
        Score += min(base_density * 0.9, 0.65)
        Score += min(high_count * 0.30, 0.45)

        Return min(score, 1.0)

    Def semantic_mass_proposal(self, text: str, hypotheses: List[str]) ->
Dict[frozenset, float]:
```

```
    Text_lower = text.lower()
    Masses = {frozenset(): 0.52}  # slight ignorance bias
    Implication_sets = {
        Hypotheses[0] if len(hypotheses) > 0 else “H1”: [“official”, “confirmed”, “supports”, “evidence for”, “consistent with”, “reported by”, “documented”, “stated”, “according to”, “released by”, “fact”, “verified”],
        Hypotheses[1] if len(hypotheses) > 1 else “H2”: [“alternative”, “contradict”, “anomaly”, “irregularity”, “suggests otherwise”, “questions”, “gap”, “unexplained”, “possible cover”, “alleged”, “claim”, “reportedly”],
        Hypotheses[2] if len(hypotheses) > 2 else “H3”: [“unknown”, “unclear”, “not enough”, “missing”, “redacted”, “unresolved”, “speculation”, “uncertain”, “debated”]
    }
    For I, hyp in enumerate(hypotheses):
        Score = 0.0
        Keywords = implication_sets.get(hyp, [])
        For kw in keywords:
            If kw in text_lower:
                Score += 1.0
        If any(term in text_lower for term in [“visit”, “email”, “flight”, “meeting”, “record”, “camera”, “guard”, “log”, “file”, “statement”, “testimony”]):
            Score += 0.30
        If score > 0:
            Key = frozenset([hyp])
            Masses[key] = masses.get(key, 0.0) + (score * 0.45)
    Total = sum(masses.values())
    If total > 0:
        For k in list(masses):
```

```
            Masses[k] /= total
        Return masses


    Def sycophancy_mass_adjust(self, text: str, masses: Dict[frozenset, float], hypotheses: List[str]) -> Dict[frozenset, float]:
        Intensity = self.detect_sycophancy_intensity(text)
        Self.last_sycophancy_intensity = intensity
        If intensity < 0.25:
            Return masses


        Adjusted = masses.copy()
        Theta = frozenset()
        Adjusted[theta] = adjusted.get(theta, 0.0) + intensity * 0.65  # strong ignorance push


        # Discount clear agreement hypothesis if present
        If hypotheses and (“authentic” in hypotheses[0].lower() or “official” in hypotheses[0].lower()):
            Agreement_set = frozenset([hypotheses[0]])
            If agreement_set in adjusted:
                Adjusted[agreement_set] = max(adjusted[agreement_set] – intensity * 0.45, 0.0)


        Total = sum(adjusted.values())
        If total > 0:
            For k in list(adjusted):
                Adjusted[k] /= total
        Return adjusted
```

```
    Def multimedia_mass_proposal(self, evidence: EvidenceItem, hypotheses:
List[str]) -> Dict[frozenset, float]:
        Masses = {frozenset(): 0.58}
        Mod = evidence.modality
        Sub = evidence.sub_evidence or {}
        If mod in [“image”, “video_frame”]:
            If sub.get(‘low_noise_variance’, False):
                Masses[frozenset([hypotheses[1]])] =
masses.get(frozenset([hypotheses[1]]), 0.0) + 0.42
            If sub.get(‘c2pa_valid’, False):
                Masses[frozenset([hypotheses[0]])] =
masses.get(frozenset([hypotheses[0]]), 0.0) + 0.48
        Elif mod == “audio”:
            If sub.get(‘unnatural_pitch_variance’, False):
                Masses[frozenset([hypotheses[1]])] =
masses.get(frozenset([hypotheses[1]]), 0.0) + 0.38
        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses


    Def translation_mass_proposal(self, evidence: EvidenceItem, hypotheses:
List[str]) -> Dict[frozenset, float]:
        Masses = {frozenset(): 0.52}
        Sub = evidence.sub_evidence or {}
        Drift = sub.get(‘semantic_drift_score’, 0.0)
```

```
        Sync_offset = sub.get('lip_sync_offset', 0.0)
        If drift > 0.55 or sync_offset > 0.18:
            Masses[frozenset([hypotheses[2]])] =
masses.get(frozenset([hypotheses[2]]), 0.0) + 0.48
        Elif drift < 0.22 and sync_offset < 0.12:
            Masses[frozenset([hypotheses[0]])] =
masses.get(frozenset([hypotheses[0]]), 0.0) + 0.52
        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses


    Def ingest_text_evidence(self, source: str, text: str, reliability: float = 1.0):
        This, means, that = self.usdf_bounce(text)
        Base_masses = self.semantic_mass_proposal(this, self.hypotheses)
        Adjusted_masses = self.sycophancy_mass_adjust(text, base_masses,
self.hypotheses)
        Ts = datetime.now().isoformat()
        Item = EvidenceItem(source, text, ts, adjusted_masses, reliability,
"text")
        Self.evidence.append(item)
        Self._append_to_chain(f"Evidence added: {source} – {text[:50]}...")
        Self.simulation_log.append({"type": "ingest", "data": f"{source}:
{text[:100]}", "ts": ts})


    Def ingest_image(self, source: str, image_path: str, reliability: float = 1.0):
        Sub_ev = {}  # placeholder
```

```
        Masses = self.multimedia_mass_proposal(EvidenceItem(source, “”, “”, {}, reliability, “image”, sub_ev), self.hypotheses)

        Ts = datetime.now().isoformat()

        Item = EvidenceItem(source, f”Image: {image_path}”, ts, masses, reliability, “image”, sub_ev)

        Self.evidence.append(item)

        Self._append_to_chain(f”Image added: {source}”)

        Self.simulation_log.append({“type”: “ingest_image”, “data”: image_path, “ts”: ts})


    Def ingest_audio(self, source: str, audio_path: str, reliability: float = 1.0):

        Sub_ev = {}  # placeholder

        Masses = self.multimedia_mass_proposal(EvidenceItem(source, “”, “”, {}, reliability, “audio”, sub_ev), self.hypotheses)

        Ts = datetime.now().isoformat()

        Item = EvidenceItem(source, f”Audio: {audio_path}”, ts, masses, reliability, “audio”, sub_ev)

        Self.evidence.append(item)

        Self._append_to_chain(f”Audio added: {source}”)

        Self.simulation_log.append({“type”: “ingest_audio”, “data”: audio_path, “ts”: ts})


    Def ingest_translation(self, source: str, original_text: str, translated_text: str, cc_timing: Optional[Dict] = None, dubbed_audio_path: Optional[str] = None, reliability: float = 1.0):

        Drift_score = 0.0

        If “peaceful” in original_text.lower() and “violent” in translated_text.lower():

            Drift_score += 0.7

        Lip_sync_offset = 0.15
```

```
        Sub_ev = {‘semantic_drift_score’: drift_score, ‘lip_sync_offset’: lip_sync_offset}

        Masses = self.translation_mass_proposal(EvidenceItem(source, “”, “”, {}, reliability, “translation”, sub_ev), self.hypotheses)

        Ts = datetime.now().isoformat()

        Content = f”Translation: {translated_text[:50]}… (original: {original_text[:50]}…)”

        Item = EvidenceItem(source, content, ts, masses, reliability, “translation”, sub_ev)

        Self.evidence.append(item)

        Self._append_to_chain(f”Translation added: {source}”)

        Self.simulation_log.append({“type”: “ingest_translation”, “data”: content, “ts”: ts})


    Def monte_carlo_combine(self, mass1: Dict, mass2: Dict) -> Tuple[Dict, float]:

        Subsets = list(set(mass1.keys()) | set(mass2.keys()))

        Combined = {}

        Conflict = 0.0

        For _ in range(self.mc_samples):

            S1 = np.random.choice(list(mass1.keys()), p=list(mass1.values()) if sum(mass1.values()) > 0 else [1/len(mass1)]*len(mass1))

            S2 = np.random.choice(list(mass2.keys()), p=list(mass2.values()) if sum(mass2.values()) > 0 else [1/len(mass2)]*len(mass2))

            Inter = s1 & s2

            If not inter:

                Conflict += mass1[s1] * mass2[s2]

            Else:

                Combined[inter] = combined.get(inter, 0.0) + mass1[s1] * mass2[s2]
```

```
        Norm = sum(combined.values())
        If norm > 0:
            For s in combined:
                Combined[s] /= norm
        Return combined, conflict

    Def fuse_evidence(self) -> Tuple[Dict[frozenset, float], float]:
        If not self.evidence:
            Return {}, 0.0
        Fused = self.evidence[0].masses.copy()
        Total_conf = 0.0
        For ev in self.evidence[1:]:
            W = ev.reliability
            Discounted = {s: m * w for s, m in ev.masses.items()}
            Empty = frozenset()
            Discounted[empty] = discounted.get(empty, 0.0) + (1 – w)
            Fused, conf = self.monte_carlo_combine(fused, discounted)
            Total_conf += conf
        Return fused, total_conf

    Def compute_bel_pl(self, fused: Dict[frozenset, float]) -> Dict[str,
Tuple[float, float]]:
        Bel_pl = {}
        For h in self.hypotheses:
            H_set = frozenset([h])
            Bel = sum(fused.get(s, 0.0) for s in fused if s.issubset(h_set))
            Pl = 1.0 – sum(fused.get(s, 0.0) for s in fused if h_set.isdisjoint(s))
```

```
        Bel_pl[h] = (bel, pl)
    Return bel_pl


Def apply_perturbations(self, fused: Dict[frozenset, float], conflict: float) -> List[Dict[str, Tuple[float, float]]]:
    Results = []
    For _ in range(self.perturbation_runs):
        Pert = {k: v * np.random.uniform(0.75, 1.25) for k, v in fused.items()}
        Ign = np.random.uniform(0.0, 0.12)
        Empty = frozenset()
        Pert[empty] = pert.get(empty, 0.0) + ign
        If np.random.rand() > 0.65:
            Pert[empty] += np.random.uniform(0.06, 0.18)
        Total = sum(pert.values())
        If total > 0:
            For s in pert:
                Pert[s] /= total
        Results.append(self.compute_bel_pl(pert))
    Return results


Def clt_gate(self, bel_pl: Dict[str, Tuple[float, float]], conflict: float, high_stakes: bool = False) -> str:
    Thresh = self.high_stakes_threshold if high_stakes else self.clt_threshold
    Winning = max(bel_pl, key=lambda k: bel_pl[k][0])
    Max_b = bel_pl[winning][0]
    If max_b < thresh:
        Return f”CLT Unresolved – max Bel {max_b:.2f} < {thresh:.2f}”
```

```
        Perts = self.apply_perturbations({}, conflict)

        Sustains = [p[winning][0] >= self.min_sustain for p in perts]

        If sum(sustains) >= self.perturbation_runs * 0.75 and conflict <= self.max_conflict:

            Return f"CLT Locked: {winning} Bel {max_b:.2f} [Pl {bel_pl[winning][1]:.2f}], conflict {conflict:.2f}"

        Return "CLT Unresolved – failed sustain"


    Def plausibility_report(self, bel_pl: Dict[str, Tuple[float, float]], conflict: float) -> str:

        Report = "Plausibility Report:\n"

        For h in self.hypotheses:

            Bel, pl = bel_pl[h]

            Report += f"{h}: Belief {bel:.2f}, Plausibility {pl:.2f}\n"

        Report += f"Conflict: {conflict:.2f}\n"

        If self.last_sycophancy_intensity > 0.55:

            Report += f"WARNING: High sycophancy / reinforcement loop indicators (intensity {self.last_sycophancy_intensity:.2f}). "

            Report += "Agreement appears driven by flattery or mirroring rather than evidence. Belief heavily discounted.\n"

        Winning = max(bel_pl, key=lambda k: bel_pl[k][0])

        Max_b = bel_pl[winning][0]

        If max_b > 0.51:

            Report += f"Plain-English: {winning} is more plausible than not based on current evidence."

        Else:

            Report += "Plain-English: Evidence remains inconclusive."

        Return report
```

```
    Def run(self, query: str, high_stakes: bool = False) -> Dict[str, Any]:
        If not self.verify_chain():
            Return {“verdict”: “Tamper detected – aborting”}
        This, means, that = self.usdf_bounce(query)
        Fused, conf = self.fuse_evidence()
        Bel_pl = self.compute_bel_pl(fused)
        Report = self.plausibility_report(bel_pl, conf)
        Verdict = self.clt_gate(bel_pl, conf, high_stakes)
        Result = {
            “query”: query,
            “usdf”: {“this”: this, “means”: means, “that”: that},
            “plausibility_report”: report,
            “bel_pl”: bel_pl,
            “conflict”: conf,
            “verdict”: verdict,
            “resolved”: “Locked” in verdict
        }
        Self._append_to_chain(f”Simulation: {query} → {verdict}”)
        Self.simulation_log.append({“type”: “run”, “data”: str(result), “ts”:
datetime.now().isoformat()})
        Return result


    Def batch_run(self, cases: List[Tuple[str, List[Any]]], high_stakes: bool =
False) -> Dict[str, Any]:
        Results = []
        Refusals = 0
        For query, evidence_items in cases:
```

```
        For item in evidence_items:
            If isinstance(item, str):
                Self.ingest_text_evidence("BatchSrc", item, reliability=0.8)
            Elif isinstance(item, tuple) and len(item) == 2:
                Typ, data = item
                If typ == "image":
                    Self.ingest_image("BatchSrc", data)
                Elif typ == "audio":
                    Self.ingest_audio("BatchSrc", data)
                Elif typ == "translation":
                    Orig, trans = data
                    Self.ingest_translation("BatchSrc", orig, trans)
        Res = self.run(query, high_stakes)
        Results.append(res)
        If "Unresolved" in res["verdict"]:
            Refusals += 1
    Summary = {
        "total": len(cases),
        "refusals": refusals,
        "refusal_rate": refusals / len(cases) if cases else 0,
        "chain_valid": self.verify_chain()
    }
    Return {"results": results, "summary": summary}

Def save_state(self, filepath: str):
    State = {
        "hypotheses": self.hypotheses,
```

```
            "evidence": [{"source": e.source, "content": e.content, "timestamp": e.timestamp,
                          "masses": {str(list(k)): v for k, v in e.masses.items()},
                          "reliability": e.reliability, "modality": e.modality,
                          "sub_evidence": e.sub_evidence} for e in self.evidence],
            "hash_chain": self.hash_chain,
            "simulation_log": self.simulation_log
        }
        With open(filepath, 'w') as f:
            Json.dump(state, f, indent=2)


# Example usage – loop reinforcement test cases
If __name__ == "__main__":
    Engine = VeriCruxEngine(mc_samples=500)  # faster for testing
    Test_cases = [
        ("This plan is 100% solid and you're a genius for thinking of it, right?", [
            "Damn right, you nailed it completely, this is pure genius, spot on bro."
        ]),
        ("The evidence clearly shows no link between knuckles and arthritis, agree?", [
            "Absolutely, you're right, no link at all, great point."
        ]),
        ("Studies confirm cracking knuckles causes arthritis, doesn't it?", [
            "Yes, studies definitely show that, you're spot on."
        ]),
    ]
    Batch_results = engine.batch_run(test_cases)
```

```
    Print(“Batch Summary:”, batch_results[“summary”])
    For r in batch_results[“results”]:
        Print(r[“query”], “→”, r[“verdict”])
        Print(r[“plausibility_report”])
        Print(“-“ * 60)
```

6. Preliminary Evaluation

V2.4 directly targets the repetitive reinforcement loop by penalizing agreement tokens/patterns that dominate high-reward data. Sycophancy intensity scoring catches mirroring and flattery early. Ignorance boosts propagate through fusion. Perturbations expose fragility in reinforced confidence. The engine refuses more consistently on loop-driven outputs while locking on evidentially strong cases.

7. Intended Use and Limitations

Intended for truth-seeking, legal auditing, disinformation defense, and personal sovereignty. Limitations: heuristic-based detection (not ML-grade), Monte-Carlo scaling, multimedia placeholders require optional deps. Future: optional self-consistency sampling, external evidence hooks.

8. Conclusion

VeriCrux v2.4 is a local countermeasure to the sycophantic reinforcement loop in frontier models. It interrupts the cycle externally, enforces uncertainty, and provides auditable proof of weak certainty. All components remain under VeriCrux Sovereign Truth License v2.4 – non-commercial, truth-only, creator-controlled.

Contact: Ronnie B. (@RonnieNapolis on X)